The Application Log

The application log is a common component. There is one application log class per process. This component encapsulates a message and a log object, providing a convenient, singleton for application level logging and message facility access. In addition, this facility provides methods for logging a message or an exception.

The Log is a common component. This facility provides methods for maintaining an in-storage, wrap-around log cache that may be configured to dump on demand or dump automatically. The number of log cache slots may also be configured. The size of a slot is arbitrary, based solely on contents; there is no mechanism for configuring the log by size.

The Log Observer is a common component. A log observer is used in conjunction with a log to facilitate automated notification. Typically, a log observer is a writer that receives the contents of the log buffer when the log is switched.

Services

The services provided by the application log are the union of the services it encapsulates.

Application Log

getLog

This method returns a reference to the application log object.

getMsg

This method returns a reference to the application message object.

logException

This method creates a log entry for an exception. The log entry consists of these elements: the localized message, the exception, and a stack trace from the exception. The message and exception text are returned to the caller.

logMsg

These polymorphic methods allow the caller to insert messages into the log.

Log

The log implements the java.util.Observable interface.

addLogEntry

These polymorphic methods are used to add entries to the log cache.

clearLog

This method flushes the log cache. Log observers will receive the log entries.

getSlots/setSlots

These attribute methods control the size of the log cache. The default number of slots is 50. This means that a log observer will be notified once for every 50 log entries.

writeLog

These polymorphic, convenience methods can be used by log observers to write log entries to a stream.

LogObserver

The log observer implements the java.util.Observer interface. The log observer provides constructors for streams and writers. Once constructed, it does not require further attention.

Properties and capabilities

The properties and capabilities facility (property facility) is part of the common utility component. Properties and capabilities act like java.util.Properties and java.util.PropertyResourceBundles with these notable additions:

- Capability values are retained as objects, rather than strings.
- Property and capability values can be secured.
- Properties and capabilities have associated metadata, providing a mechanism for applications to use and manipulate them without a priori knowledge.
- Properties and capabilities may be hierarchically structured constructs, e.g., the logon property is a structured property whose default attributes are user and password.
- Property and capability values may be computed on the fly, e.g., the value may be retrieved from the data source or represent a virtual or class attribute.
- Properties and capabilities may be associated with user dialogs.

Capabilities are read-only properties. They are often static attributes of the underlying data source, e.g., the maximum length of a command or the maximum number of columns in a select statement. They may also be attributes of a driver, e.g., a driver supports the procedure interface or a driver supports concurrent operations within a connection. Properties and capabilities are composed of value and metadata components. The value and metadata components are stored in property resource bundles. Both the

values and the metadata component attributes may be localized. Get and set accessor methods are provided for properties and capabilities. Applications should not use the capabilities set accessor methods. Drivers use these methods to reflect the capabilities.

Locating Properties and Capabilities

All property facility property resource bundles are located in the *properties* directory. The parent directory of the *properties* directory is specified in the CLASSPATH. The java.util.ResourceBundle *getBundle()* method is used to load the property facility property resource bundles. The property facility uses a distinct naming scheme to insure that class name collisions do not occur. This scheme facilitates having the property resource bundles in one directory.

To retrieve a property resource bundle, two pieces of information are required: the fully qualified base name of the bundle and a locale identifier. These two strings are concatenated together, separated by an underscore to form a class name. An attempt is then made to load a class with that name using the default system loader. If a class with the name cannot be loaded, then the name is successively shortened until a resource bundle class is successfully loaded and instantiated. The *getBundle()* method looks for a property resource bundle whenever a class name fails to produce a resource bundle object. In particular, the class name is appended with the string ".properties". If such a file exists, a PropertyResourceBundle object is created for that properties file.

The naming scheme for the property facility is:

- All property resource bundles, i.e., both the properties and capabilities value and metadata files, are placed in the *properties* directory.
- The directory containing the *properties* directory is in the CLASSPATH.
- The separators ('.') of the fully qualified class name, i.e., <package name>.<class name>, are changed to underscores ('_') to produce a file name.
- The file name extension, ".properties" is used to complete the file name.

For example, the properties and capabilities file names for the class com.sqribe.access.DataSource would be:

com_sqribe_access_DataSource_Properties.properties

This property resource bundle contains property attribute values for the DataSource class.

com_sqribe_access_DataSource_PropertyDescriptions.properties

This property resource bundle contains property attribute metadata for the DataSource class.

com_sqribe_access_DataSource_Capabilities.properties

This property resource bundle contains capability attribute values for the DataSource class.

com_sqribe_access_DataSource_CapabilityDescriptions.properties

This property resource bundle contains capability attribute metadata for the DataSource class.

These files would be placed in the *properties* directory. Scanning through the *properties* directory, you will notice that many of the properties and capabilities refer to the DataSource class. The com.sqribe.access.DataSource interface is the base for all driver DataSource implementations. Put another way, the DataSource interface provides the basic properties and capabilities for data sources. The DataSourceAdapter provides default processing for data sources. Each driver extends the DataSourceAdapter and may have additional or overriding properties and capabilities. The DataSourceAdapter creates a property sheet for the data source. To create the property sheet, it generates a copy of the basic properties and capabilities and then augments the copy with the driver specific properties and capabilities. We say, the driver data source inherits the basic data source properties and capabilities.

A driver connection uses a copy of the properties and capabilities for the data source. This is done in an analogous manner to data source property sheet creation. There is a Connection interface, a ConnectionAdapter, and each driver implements a connection object that extends the ConnectionAdapter. The driver connection objects may provide specific properties and capabilities. This pattern is followed throughout DDO.

While the driver usage pattern is different than that used for the message facility, the property resource bundle processing for the property facility is the same. Java encourages the use of packages. Packages can be viewed as autonomous components. The classes within a package form a close knit set of relationships. These relationships are often exposed through inheritance and interface relationships. Further, component users also extend classes or implement interfaces prescribed in a package. The property facility

recognizes these relationships and uses them to locate properties and capabilities metadata and values. Let's look at the flow for establishing the properties and capabilities for a connection to an Oracle database through the DDO relational database driver.

- The data source class name of the DDO relational database driver, com.sqribe.jdbcacc.JDBCDataSource, is used as the basis for locating properties and capabilities.
- The property facility descends the interface graph for this class.
- Once at the leaf, the facility descends the inheritance graph for the leaf class.
- Once at the inheritance leaf, the facility obtains the property resource bundle for the leaf class.
- The facility ascends the inheritance graph, merging attributes from the property resource bundles.
- Once at the root of the inheritance graph for the leaf in, the facility ascends to the interface graph, performing the inheritance graph merge for each interface.
- After completing the class interface merge processing, the facility moves to the inheritance graph.
- Again, a complete descent is made, and a bottom-up attribute merge is performed. Note, an inherited class may implement interfaces; hence, interface merge processing occurs while ascending the inheritance graph.
- Finally, the attributes contained in the com.sqribe.jdbcacc.JDBCDataSource are merged.
- This processing is performed for each of the four property resource bundle types, i.e., properties, property descriptions, capabilities, and capability descriptions. The result is a new property sheet, used for this data source.
- At this point the driver may merge data source specific attributes. In our example, the driver would be merging attributes specific to an Oracle data source. This is not done for the DDO relational database driver.
- The application may alter or augment the properties contained in the new property sheet. The modifications are scoped to the given property sheet. In this example, the data source is associated with a specific Oracle database instance. It may, for example, be reasonable to use a single user name and password for all connections to this data

source for the life of this DataSource object. In this case these logon properties would be set once and used for all connections, i.e., all connections would be opened using the same user name and password properties.

- On the other hand, the application may need a different user name and password for each connection. This is done by setting the user name and password properties prior to opening a specific connection.
- What makes both of these scenarios possible, is the copy of the property sheet made for the connection instance. This way property sheet changes made through the connection object do not affect the data source object and vice versa.
- The DDO relational driver supports concurrent operations on a given connection, when the underlying relational database supports this level of concurrency. The property sheet for the connection is shared by all threads. Hence, changes to the connection property sheet will be observed by all threads.

Given a class, say com.sqribe.jdbcacc.JDBCDataSource, in a driver, the properties facility will fabricate a property resource bundle file name from this fully qualified class name. The fabricated name is given to the resource bundle to locate a property resource bundle. If the property resource bundle was located, then a property adapter is used to merge the contents of the bundle into the given property container. The StringPropertyAdapter is used to merge properties. The ObjectPropertyAdapter is used to merge capabilities. These adapters implement the PropertyAdapter interface. The ObjectPropertyAdapter creates objects for attribute values, while the StringPropertyAdapter uses strings for attribute values.

Let's follow the property description merge for the class, com.sqribe.jdbcacc.JDBCDataSource.

1. Obtain the class name.
2. Descend the implementation graph for the class:
   - Look in the class, com.sqribe.jdbcacc.JDBCacc
   - Look in the class, com.sqribe.access.Access
   - Look in the class, com.sqribe.comutil.Util
   - Merge property descriptions while ascending this graph.
3. Descend the inheritance graph for the class:

- Look in the class, com.sqribe.access.DataSourceAdapter
- Look in the class, com.sqribe.access.DataSource
- Look in the class, com.sqribe.access.Access
- Look in the class, com.sqribe.comutil.Util
- Merge property descriptions while ascending this graph.

Each time a class is considered for a property merge, the class name is converted to a property resource bundle name. In this example, com.sqribe.jdbcacc.JDBCacc would be converted to properties.com_sqribe_jdbcacc_JDBCacc_PropertyDescriptions*. The asterisk ('*') in the name indicates where the resource bundle search appends the localization suffix.

Let's look at the affect of adding a French localization (without the country identifier) to the property search. For this localization, we have translated the logon property descriptions. When we instantiate the DDO relational database driver, these property descriptions are merged into the property sheet.

1. com_sqribe_access_DataSource_PropertyDescriptions
2. com_sqribe_access_DataSource_PropertyDescriptions_fr
3. com_sqribe_jdbcacc_JDBCDataSource_PropertyDescriptions
4. com_sqribe_jdbcacc_JDBCDataSource_PropertyDescriptions_fr

The French property description files extend the default property description files, supplying in this case localized description text. This algorithm is different from simple java.util.ResourceBundle processing, where a localized resource bundle represents a complete replacement of the less specific resource bundle. Here, the localized resource bundle extends the less specific bundle.

Property Sheets

So, far we have discussed properties, capabilities, and property sheets, but have not described their usage. Let's focus on the how to access property sheets. A PropertySheet encapsulates the behaviors for properties, capabilities and their descriptions. A PropertySheet is a composite object. The secure property and capability descriptions and values are accessible through a property sheet.

Properties, capabilities, and their descriptions are stored in property resource bundles. These files are relative to the directory containing the class whose properties or

capabilities were requested. Specifically, they are located in the directory "properties" which is in the directory where the class (or jar) was loaded. The resource bundle path name is of the form:

properties/<package name with '.' replaced by '_'>_<class name>_<type>.properties

For example, the property file for com.sqribe.access.DataSource would be:

properties/com_sqribe_access_DataSource_Properties.properties

Where:

com_sqribe_access is the package name,

DataSource is the class name, and

Properties is the type.

The types are:

Properties

These are the property values for the given class. Properties are configurable attributes of a feature or facility.

Property Descriptions

These are the property descriptions for the given class. Property descriptions define the characteristics of a property. There is sufficient information in a description to formulate and syntactically validate a property. That is, the description provides information and classes for explaining the use of a property and methods to create and validate the property value.

Capabilities

These are the capability values for the given class. Capabilities are (read only) configured attributes of a feature or facility.

Capability Descriptions

These are the capability descriptions for the given class. Capability descriptions define the characteristics of a capability. There is sufficient information in a description to formulate and syntactically validate a capability. That is, the description provides

information and classes for explaining the use of a capability and methods to create and validate the capability value.

Methods

These are the property sheet instance methods available to applications.

copy

Merge the given property sheet into the current property sheet.

getProperty

Retrieve the named property value.

setProperty

Change or add a property value. While getProperty takes the property name as the key, this polymorphic method takes either the property description object or the property name as the key. This allows property descriptions to be added on the fly.

getPropertyNames

Retrieve an enumeration of the property names. This list is derived from the property values, rather than the property descriptions, container.

getPropertyDescription

Retrieve the named property description.

setPropertyDescription

Add or replace a property description with the given property description.

getPropertyDescriptionNames

Retrieve an enumeration of the property description names.

getCapability

Retrieve the named capability. While retrieving a property value will return a string or null, retrieving a capability will return an object or null.

getCapabilityNames

Retrieve an enumeration of the capability names. This list is derived from the capability values, rather than the capabilities descriptions, container.

getCapabilityDescription

Retrieve the named capability description.

getCapabilityDescriptionNames

Retrieve an enumeration of the capability description names.

Secure properties and capabilities

A distinguishing characteristic of this common utility component is the encryption of secret values. One of the attributes given in a property (or capability description) is whether the associated value should be secured. If so, the value is retained in an encrypted form. When a get accessor method requests the attribute, a decrypted copy is made. The decrypted copy should be a temporary (local) variable, discarded when the containing block goes out of scope.

Property descriptions

A PropertyDescription describes the attributes of a property. Property descriptions are common to both property and capability metadata. A property may be required or optional. The property has a name, which is its key. A property has a descriptive name. This is a short explanation of the property that may be appropriate for a tooltip. A property takes a value or value set. A property value may be secured, e.g., a passphrase. This means that the value is stored in memory and transferred in an encrypted form. These values are always passed as Strings. Values are expressed as type specific Java objects, e.g., java.lang.Boolean. A value may have domain requirements in the form of a range or list. The value may be indexed, i.e., may have multiple values, transferred as an array of objects. A value may have an associated description. Hence, indexed values may have indexed descriptions.

The design goal for the property facility is to eliminate the need for a client to have a priori knowledge of a data source driver to establish driver properties. That is, the client may present the property descriptions to the user to complete. The description content should contain sufficient information to present and validate a property.

Attribute descriptions and use:

Name	Description	Re-quired	Explanation
Name	The fully qualified name of the property	Yes	
Description	The descriptive name of the property (e.g., help, tooltip)	No	
ClassName	The class name of the value type, e.g., Integer, Boolean	No	Must agree with indices
AuxProc	The class name of the class	No	Must agree with indices

	providing virtual fetch and post processing of a property or capability. This class implements the PropertyAuxProc interface.		
Required	Whether a value is required for this property	No	Default: not required
Secured	Whether the value for this property must be encrypted	No	Default: not secured
ValidationType	The type of validation: none, range, or list	No	Default: no validation
ValidationValues	The validation values: none:null; range: 0(min), 1(max); list: discrete values. Note: values are specified as strings and are converted by the validator to the proper type	No	Must agree with validation type
Validator	The class name of the validation implementation of the interface, PropertyValidator. **See Also** PropertyValidator	No	Default: no validator
Indices	The property descriptions of the indexed values. When there are indexed values, then classname is ignored; otherwise classname must be specified	No	Must be consistent with validation values
Dialog	The class name of a custom property dialog to display (and, optionally, validate) this property and its property subtree.	No	Default: no custom dialog

The attribute keys are scoped names whose final component represents one of the names in the table. The only required attribute is the key with the Name extension. Other attributes are context sensitive. When the Indices extension is specified, indicating a structured property non-leaf node, then these attributes should not be specified: ClassName, AuxProc, Secured, ValidationType, ValidationValues, and Validator. For a leaf node, the ClassName is required. This indicates the type of object the value should represent. For example, ClassName with a value of java.lang.Integer, indicates that integer values (or a numeric string) will be acceptable. This provides syntactic validation of the value. Adding a ValidationType and ValidationValues, provides some simple semantic checks. A Validator may be associated with the property to perform the checks specified in the

ValidationType and ValidationValues. Validators are provided for Java primitive types and selected objects.

Property auxiliary services

The PropertyAuxProc interface is used to implement specialized runtime processing. These methods are used to obtain driver attributes (either properties or capabilities) that are not part of the property/capability resource bundles, per se; rather, these properties or capabilities are supported through some other mechanism, e.g., they may be attributes of the underlying data source that cannot (or should not) be persistent in a property/capability resource bundle.

The get and set methods of this interface are independent. The use of the get method to obtain a virtual property does not imply the use of the set method to post process the property. Likewise, employing the set method to update class attributes does not imply that the get method is used to create those attributes.

While the get method, typically, acquires the information, it should arrange for caching the value in the hash table. This implies that the get method should have some first time logic. Further, properties (and capabilities) utilizing this interface are not saved, i.e., made persistent. This implies that the get method could test for the presence of the property in the hash table as the first time logic, suffering the cost of a hash table lookup for each test.

Property validators

A PropertyValidator provides an interface to property validation classes. In addition, the AbstractPropertyValidator class implements major portions of the interface, reducing the effort to two methods: createValue and compare. The BigDecimalValidator is shown below to provide a feel for validators.

```
public class BigDecimalValidator extends AbstractPropertyValidator {

    /**
     * Create the value from a string
     * @param pValue        The value to be converted
     * @return              The converted value
     * @exception PropertyException
     *                      Generic exception indicating that the value
     *                      could not be converted
     */
    public Object createValue(String pValue) throws PropertyException {
        try {
            return new BigDecimal(pValue);
        } catch (Throwable e) {
            throw new PropertyException(e.getMessage());
        }
    }

    /**
     * Compare to values
     * @param pValue1       First comparand
     * @param pValue2       Second comparand
     * @return              First<Second==-1;
     *             First==Second==0;
     *                First>Second==1
     * @exception PropertyException
     *                      Generic exception indicating that the value
     *                      was not the correct data type
     */
    public int compare(Object pValue1, Object pValue2) throws
                       PropertyException {
        return (((BigDecimal)pValue1).compareTo((BigDecimal)pValue2));
    }
}
```

The common utilities package provides implementations for Java primitives and Java SQL types. These validators extend AbstractPropertyValidator.

- BigDecimalValidator
- BooleanValidator
- ByteValidator
- CharacterValidator
- DateValidator
- DoubleValidator
- FloatValidator
- IntegerValidator
- LongValidator
- ShortValidator

- StringValidator

Instrumentation

Diagnostic Exits

The message facility provides a mechanism for invoking diagnostic classes. A diagnostic class may be associated with a message. When the message is requested the diagnostic class is instantiated. The common utility facility has an empty interface, Diagnostic, and instructions for constructor signatures.

Driver developers are encouraged to use the Debug class in conjunction with the toString method for the classes comprising their driver package. The Debug class is a specialization of the StringBuffer class, focused upon dumping class state information. DDO makes extensive use of these classes for runtime diagnostic information.

Timing

There is a set of monitor properties that provide timing instrumentation. These are based on a simple common utility stopwatch class called Monitor. Monitor provides start and stop timing methods. The Access package includes a DataSourceMonitor class that uses this mechanism for medium level timing statistics. Data Source Monitor provides a convenience mechanism for presenting the settings for various monitoring states.

The monitoring states are set from properties when this singleton class is instantiated by the DriverSourceManager. The property descriptions for these states are updated through the DataSourceMonitorAuxProc class that is registered for the monitor properties.

It is possible to override monitoring states associated with a specific driver through the use of virtual properties.

For more information on the individual monitors refer to the monitor property descriptions.

While the monitors are setup as a hierarchy, querying the monitors is flat. Here are the relevant retrieval/loading monitoring points:

retrievalExecute

execute monitoring is active.

retrievalCall

call monitoring is active.

retrievalGetData

getData monitoring is active.

metadataSchema

schema names monitoring is active.

metadataSchemaObjects

schema objects monitoring is active.

metadataSchemaObjectColumns

schema object columns monitoring is active.

metadataSchemaProcedures

schema procedures monitoring is active.

metadataSchemaProceduresMeta

schema procedures metadata monitoring is active.

propertysheetLoad

propertysheet load monitoring is active.

drivermanagerLoad

data source manager load monitoring is active.

Having described the invention in terms of a preferred embodiment, it will be recognized by those skilled in the art that various types of general purpose computer hardware may be substituted for the configuration described above to achieve an equivalent result. Similarly, it will be appreciated that arithmetic logic circuits are configured to perform each required means in the claims for processing applications which access multiple data sources of different type; for permitting the middleware system to provide for drivers for new data sources to be added in a plug-and-play manner; and for development of new data source drivers at run-time. It will be apparent to those skilled in the art that modifications and variations of the preferred embodiment are possible, which fall within the true spirit and scope of the invention as measured by the following claims.

CLAIMS

What is claimed is:

1. A computer data access apparatus comprising:

a. a computer system having a processor, a memory, a program in said memory, a display screen and an input/output unit; and

b. a middleware mechanism residing in the computer system and configured to access a plurality of data sources of disparate type in response to input commands from a single application mechanism.

2. The apparatus of claim 1 wherein the middleware mechanism is coupled to a driver mechanism to access one or more data sources.

3. The apparatus of claim 1 further comprising a plurality of driver mechanisms coupled to the middleware mechanism, wherein each of the driver mechanisms is coupled to at least one data source.

4. The apparatus of claim 3 wherein the driver mechanism coupled to a data source can negotiate its preferences and capabilities with the single application mechanism.

5. The apparatus of claim 1 further comprising a first program code mechanism in the middleware mechanism configured to create a driver mechanism for access to a data source.

6. The apparatus of claim 5 further comprising a second program code mechanism in the middleware mechanism configured to maintain a registry of driver mechanisms and related data sources.

7. The apparatus of claim 6 further comprising a third program code mechanism in the middleware mechanism configured to maintain property sheets for each data source, the property sheets providing information concerning data source capabilities and attributes.

8. The apparatus of claim 7 further comprising a fourth program code mechanism adapted to function as a configuration interface to maintain metadata about a data source.

9. The apparatus of claim 1 wherein the plurality of data sources may be situated in different physical locations interconnected by electronic means including the Internet.

10. The apparatus of claim 1 wherein the computer system is a server computer which is remote from and electronically coupled to a client computer, the client computer providing an interface to the middleware mechanism residing in the server computer.

11. The apparatus of claim 1 wherein the middleware mechanism presents streamed result sets to the application mechanism.

12. The apparatus of claim 1 further comprising a capabilities model coupled to the middleware mechanism wherein properties and capabilities may be queried or set.

13. The apparatus of claim 12 wherein the capabilities model provides a mechanism for data source independent logon.

14. A middleware data access apparatus comprising means for accessing data in a plurality of data sources of different types from a single application code mechanism.

15. The middleware data access apparatus of claim 14 further comprising means for creating a driver means, the driver means for providing access to one or more data sources.

16. The middleware data access apparatus of claim 15 wherein the means for creating a driver means uses a pre-existing skeleton driver.

17. The middleware data access apparatus of claim 14 further comprising means for displaying data obtained from the plurality of data sources in a common format regardless of data structure of the data sources.

18. The middleware data access apparatus of claim 14 further comprising means for presenting steamed result sets to the single application code mechanism.

19. A method for using a computer for accessing data from a plurality of disparate data sources comprising the steps of:

a. providing a driver for one or more of the plurality of disparate data sources;

b. creating an application to execute commands on one or more of the disparate data sources;

c. providing a middleware data access mechanism to execute the commands and to use drivers to obtain desired data from the designated data sources, and to display the obtained data in a standard format.

20. The method of claim 19 comprising the additional steps of;

d. determining whether the middleware data access mechanism has a driver for a data source designated by the application; and

e. if not, dynamically instantiating a specification for and selecting a driver for the data source designated by the application; and

f. using the selected driver to access data from the data source designated by the application.

21. The method of claim 19 wherein the obtained data is displayed as streamed result sets.

22. The method of claim 19 comprising the additional step of providing a capabilities model wherein properties and capabilities can be queried or set.

23. The method of claim 22 wherein the capabilities model can be used to determine a logon for a data source.

24. A computer program product embodied on a computer readable medium for developing applications capable of accessing a plurality of disparate data sources comprising:

a data access middleware component that stores, retrieves and manipulates data utilizing a plurality of functions; and

a client component including:

an adapter component that transmits and receives data to/from the data access middleware component,

a user interface component that is adapted to handle events generated by a user to generate data source access commands for execution by a plurality of disparate data sources; and

wherein the data access middleware component is adapted to receive the data source access commands, determine whether drivers exists for the indicated plurality of disparate data sources, and use the drivers to execute the data source access commands, and return any results to the client component for display to the user.

25. A computer program product embodied on a computer readable medium for accessing a plurality of disparate data sources comprising;

a. a driver code segment for each of the plurality of disparate data sources;

b. an application code segment to execute commands on one or more of the disparate data sources;

c. a middleware data access code segment for executing the commands and using drivers to obtain desired data from designated data sources, and for displaying the obtained data in a standard format regardless of the source of the data.

26. The computer program product as set forth in claim 25 wherein the obtained data is displayed as streamed result sets.

27. The computer program product as set forth in claim 25 further comprising:

d. a code segment for determining whether the middleware data access code segment has a driver for a data source designated by the application code segment; and if not, dynamically instantiating a specification for and selecting a driver for the data source designated by the application code segment; and for using the created driver to access data from the data source designated by the application.

28. A computer program product embodied on a computer readable carrier wave for developing applications capable of accessing a plurality of disparate data sources comprising:

a data access middleware component that stores, retrieves and manipulates data utilizing a plurality of functions; and

a client component including:

an adapter component that transmits and receives data to/from the data access middleware component,

a user interface component that is adapted to handle events generated by a user to generate data source access commands for execution by a plurality of disparate data sources; and

wherein the data access middleware component is adapted to receive the data source access commands, determine whether drivers exists for the indicated plurality of disparate data sources, and use the drivers to execute the data source access commands, and return any results to the client component for display to the user.

29. A computer program product for developing applications capable of accessing a plurality of disparate data sources comprising:

a data access middleware component that stores, retrieves and manipulates data utilizing a plurality of functions; and

a client component including:

an adapter component that transmits and receives data to/from the data access middleware component,

a user interface component that is adapted to handle events generated by a user to generate data source access commands for execution by a plurality of disparate data sources; and

wherein the data access middleware component is adapted to receive the data source access commands, determine whether drivers exists for the indicated plurality of disparate data sources, and use the drivers to execute the data source access commands, and return any results to the client component for display to the user.

Figure 1

100 ⟹ Typical Internet Network Configuration

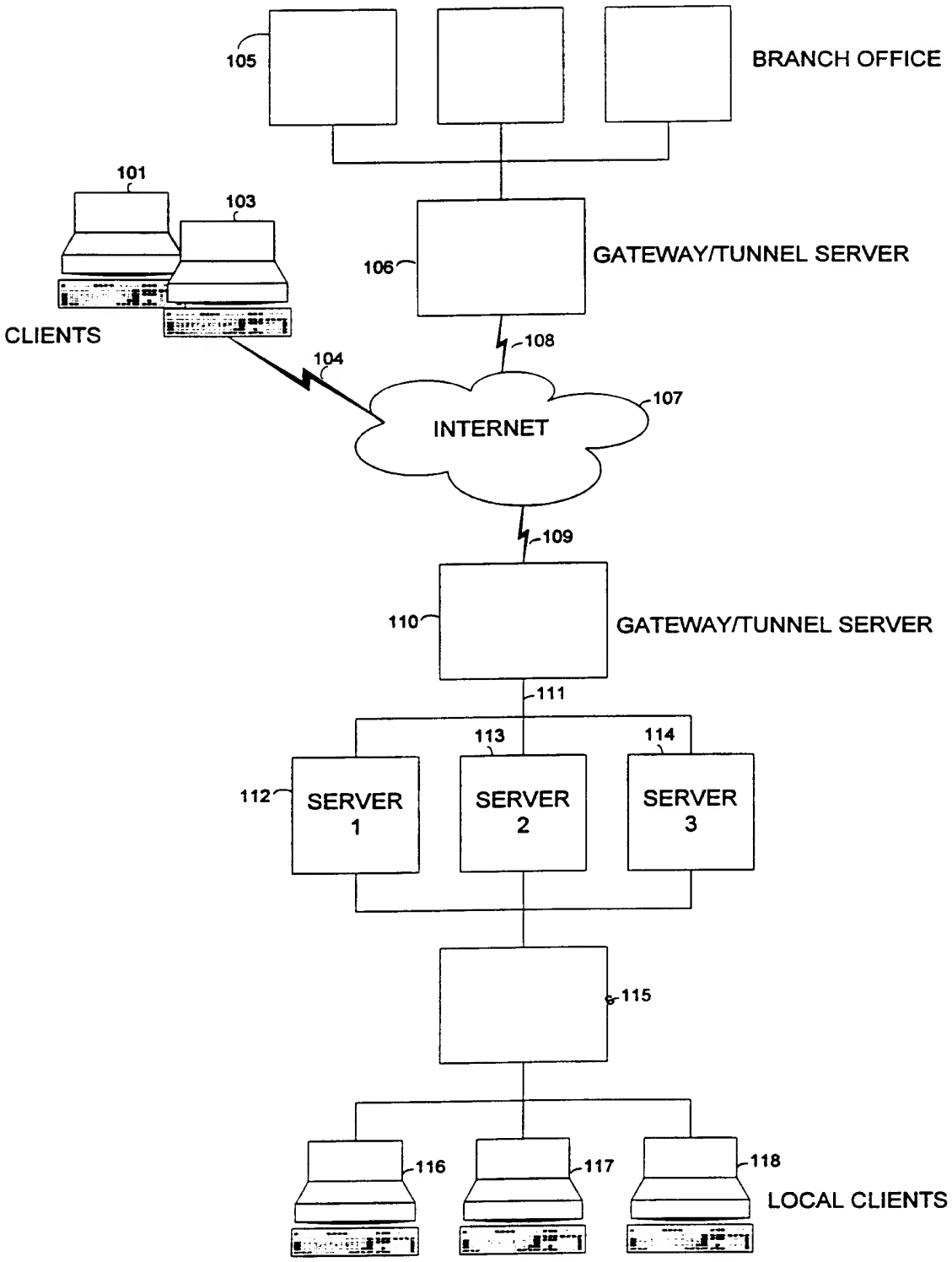
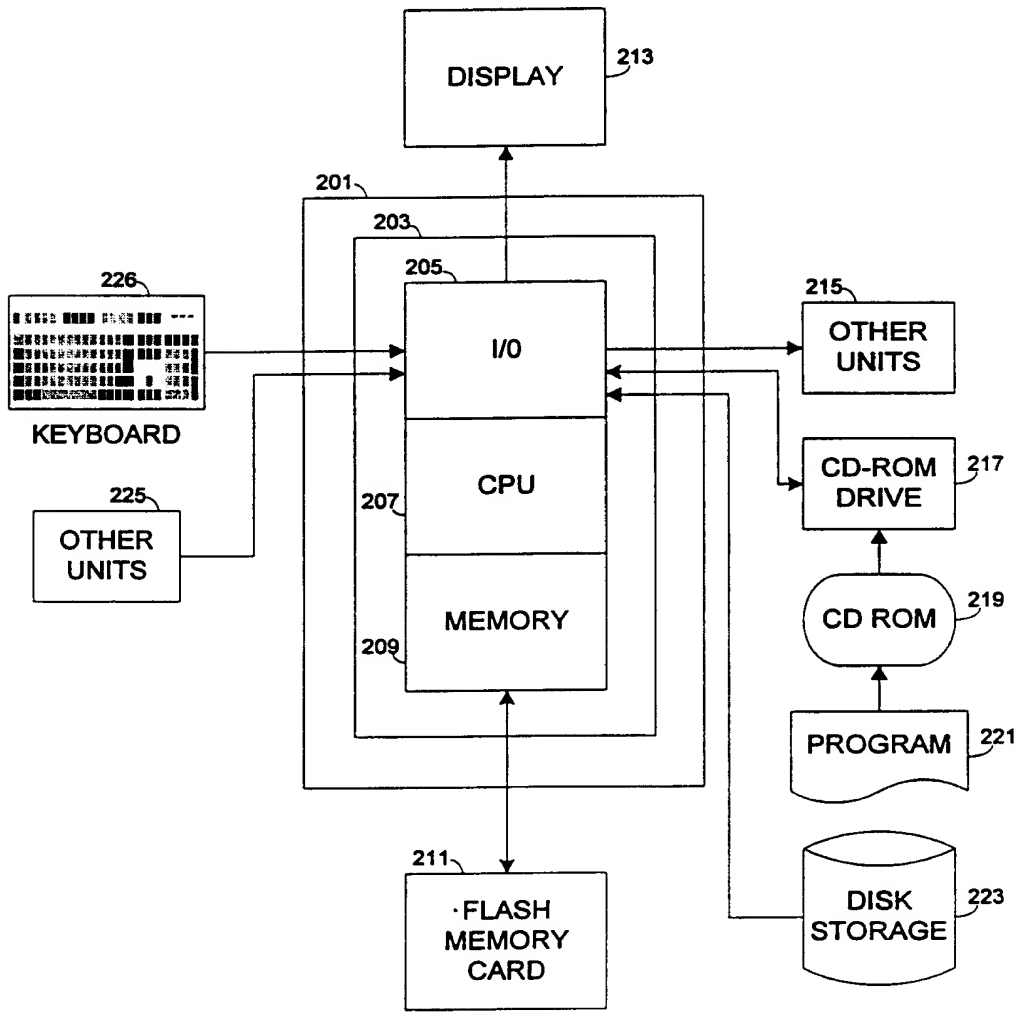


Figure 2

200 Typical General Purpose Computer/



300

FIG. 3

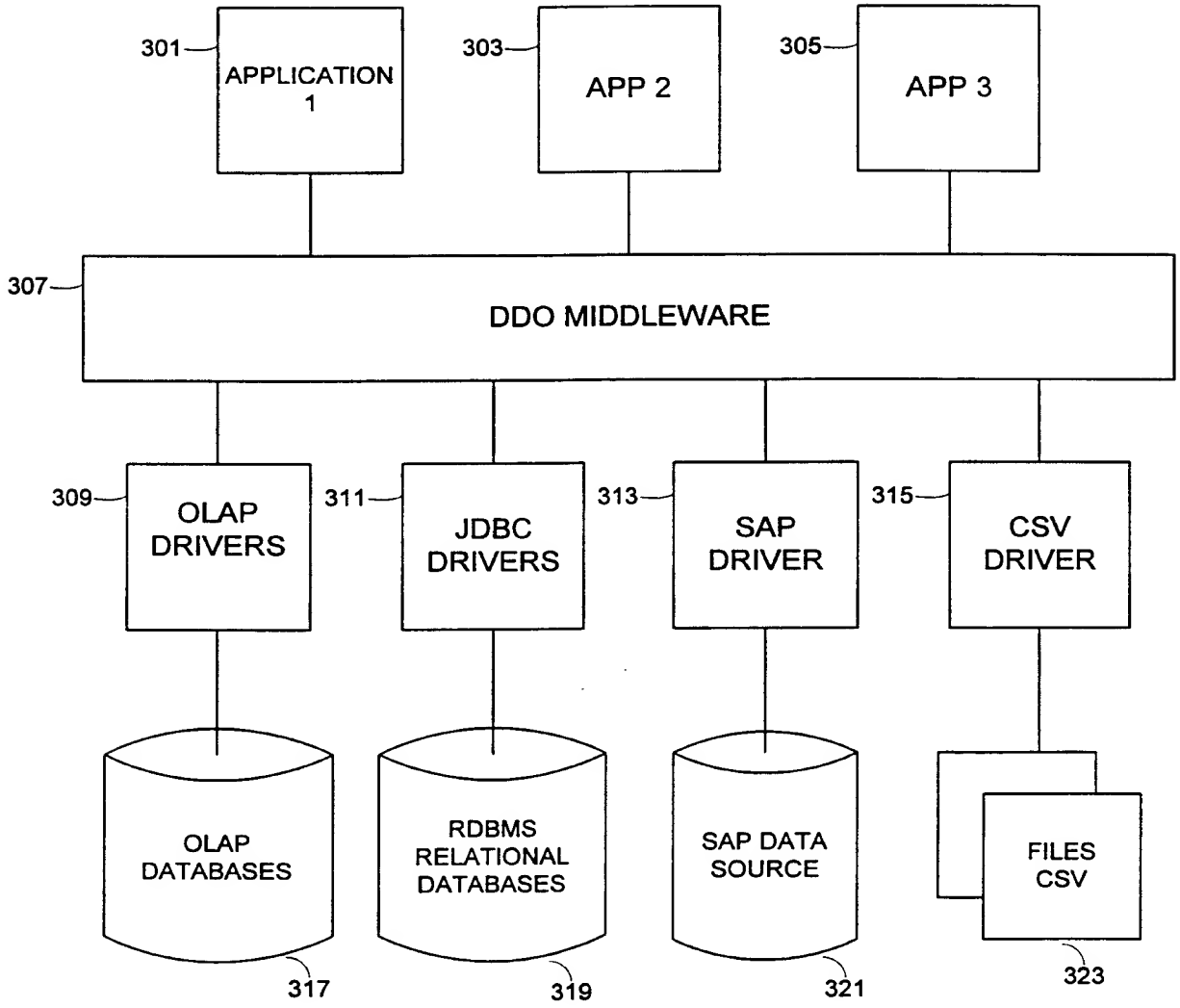
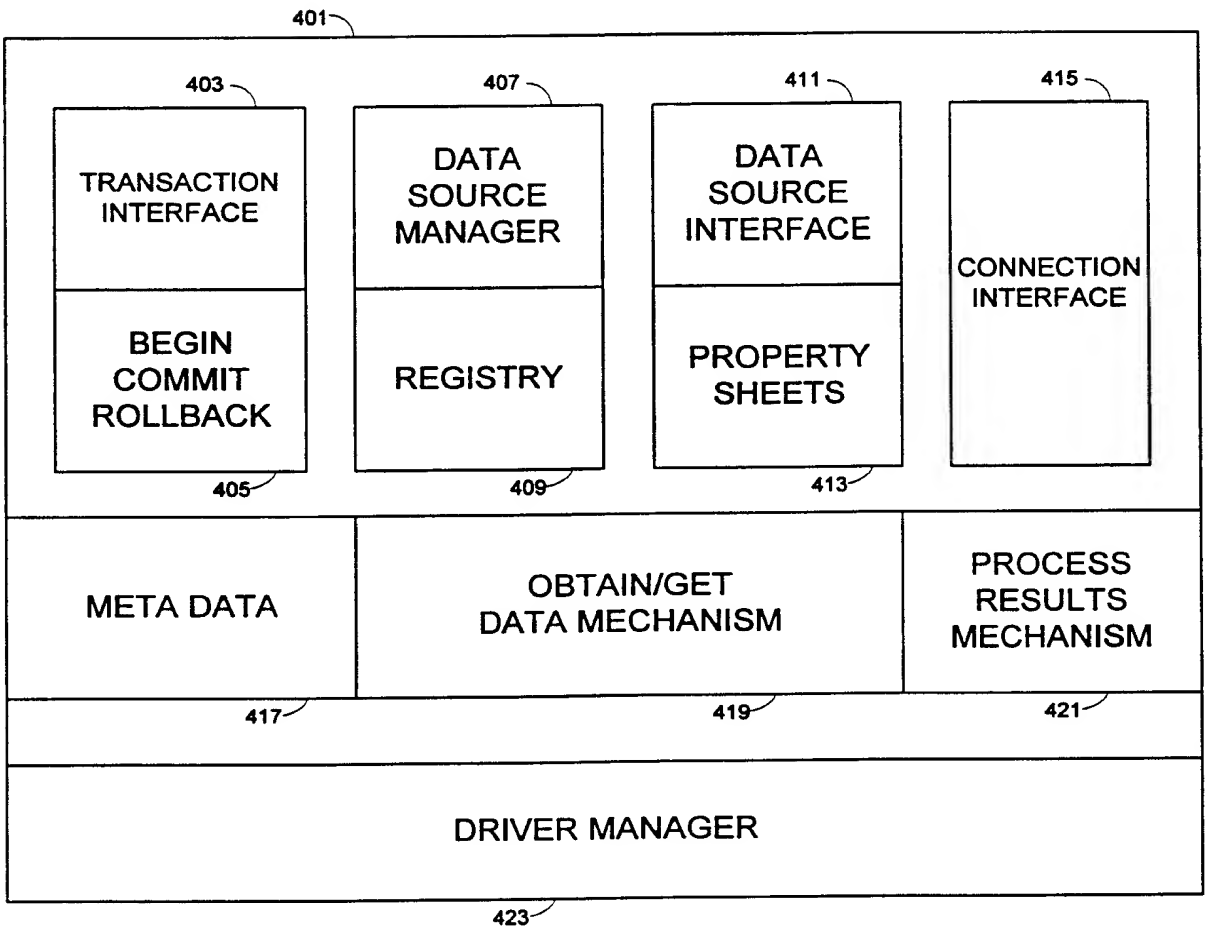
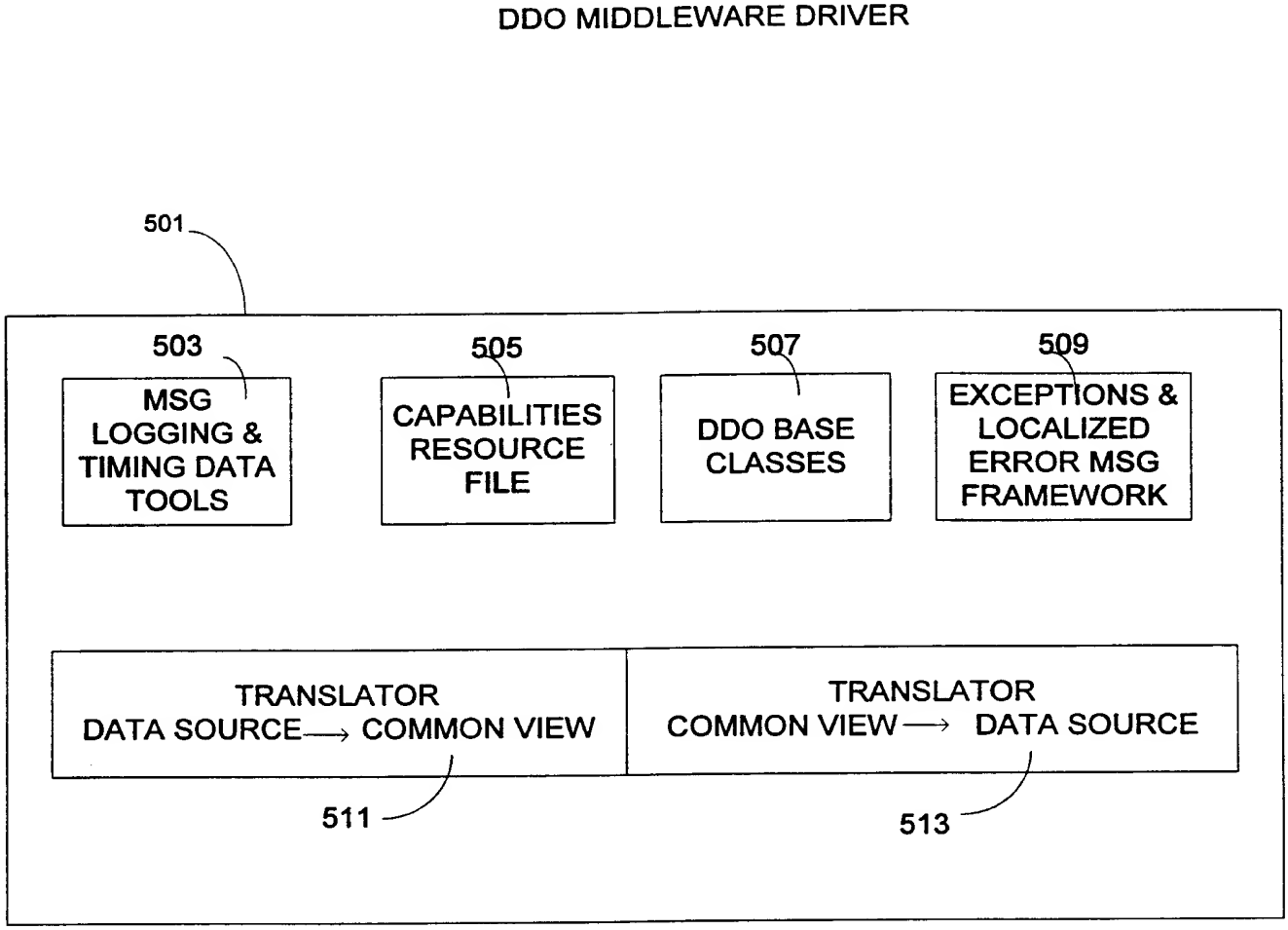

301
APPLICATION 1
303
APP 2
305
APP 3
307
DDO MIDDLEWARE
309
OLAP DRIVERS
311
JDBC DRIVERS
313
SAP DRIVER
315
CSV DRIVER
OLAP DATABASES
317
RDBMS RELATIONAL DATABASES
319
SAP DATA SOURCE
321
FILES CSV
323

FIG. 4


DDO MIDDLEWARE
401
403
TRANSACTION INTERFACE
BEGIN COMMIT ROLLBACK
405
407
DATA SOURCE MANAGER
REGISTRY
409
411
DATA SOURCE INTERFACE
PROPERTY SHEETS
413
415
CONNECTION INTERFACE
META DATA
417
OBTAIN/GET DATA MECHANISM
419
PROCESS RESULTS MECHANISM
421
DRIVER MANAGER
423

500

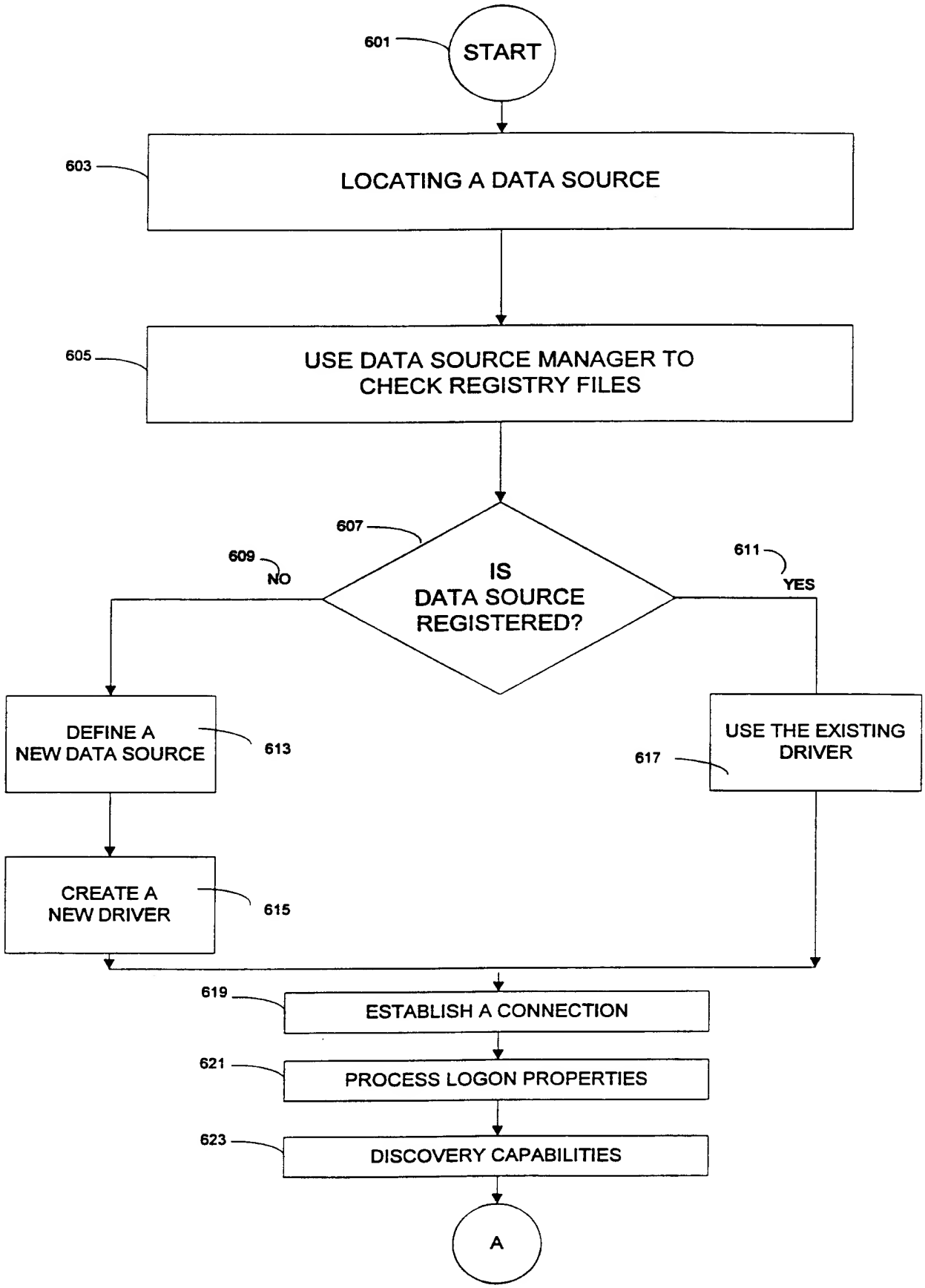
FIG. 5


DDO MIDDLEWARE DRIVER
501
503
MSG LOGGING & TIMING DATA TOOLS
505
CAPABILITIES RESOURCE FILE
507
DDO BASE CLASSES
509
EXCEPTIONS & LOCALIZED ERROR MSG FRAMEWORK
TRANSLATOR DATA SOURCE ⟶ COMMON VIEW
TRANSLATOR COMMON VIEW ⟶ DATA SOURCE
511
513

600

FIG. 6A

CREATING A DDO APPLICATION



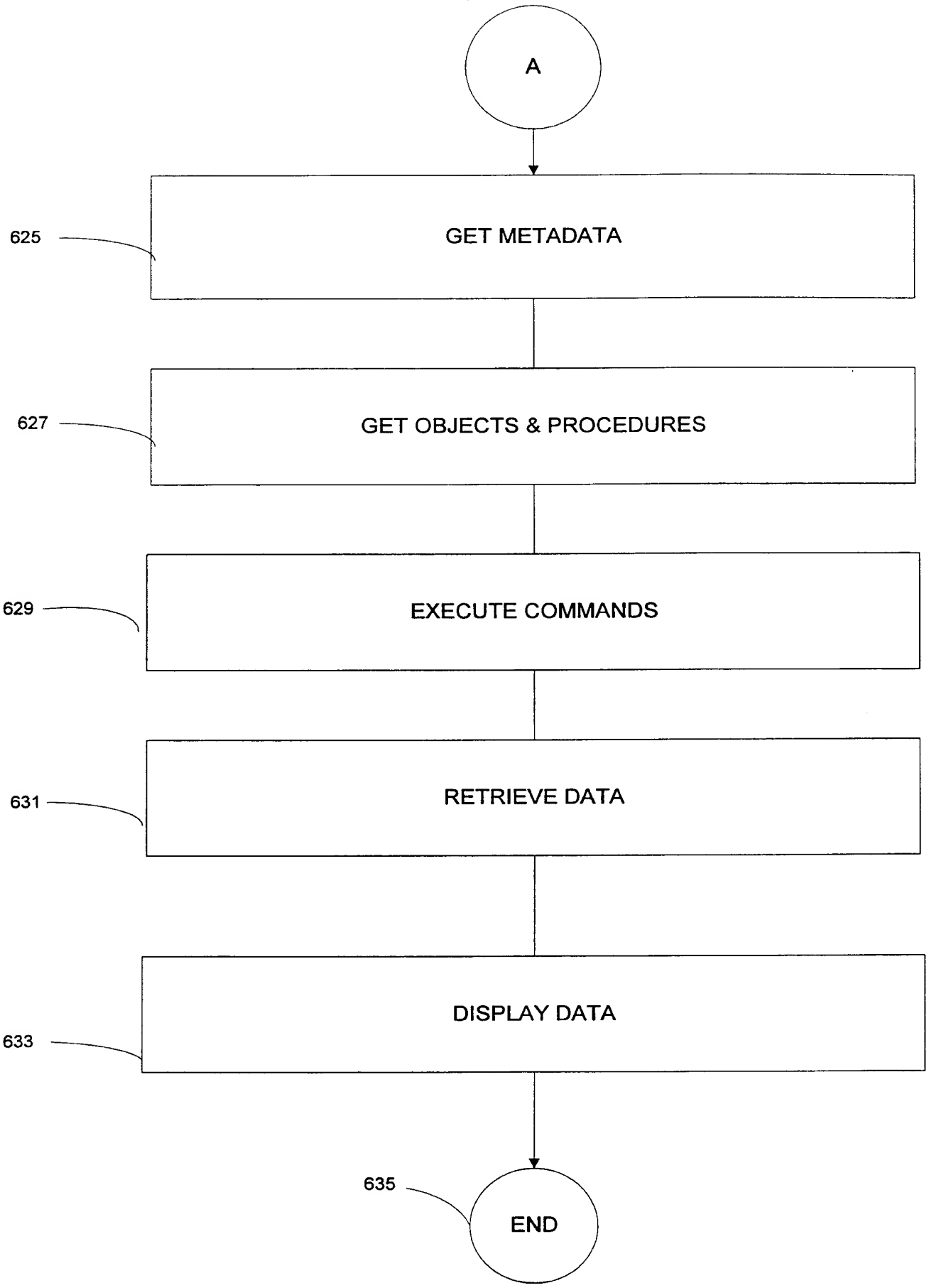
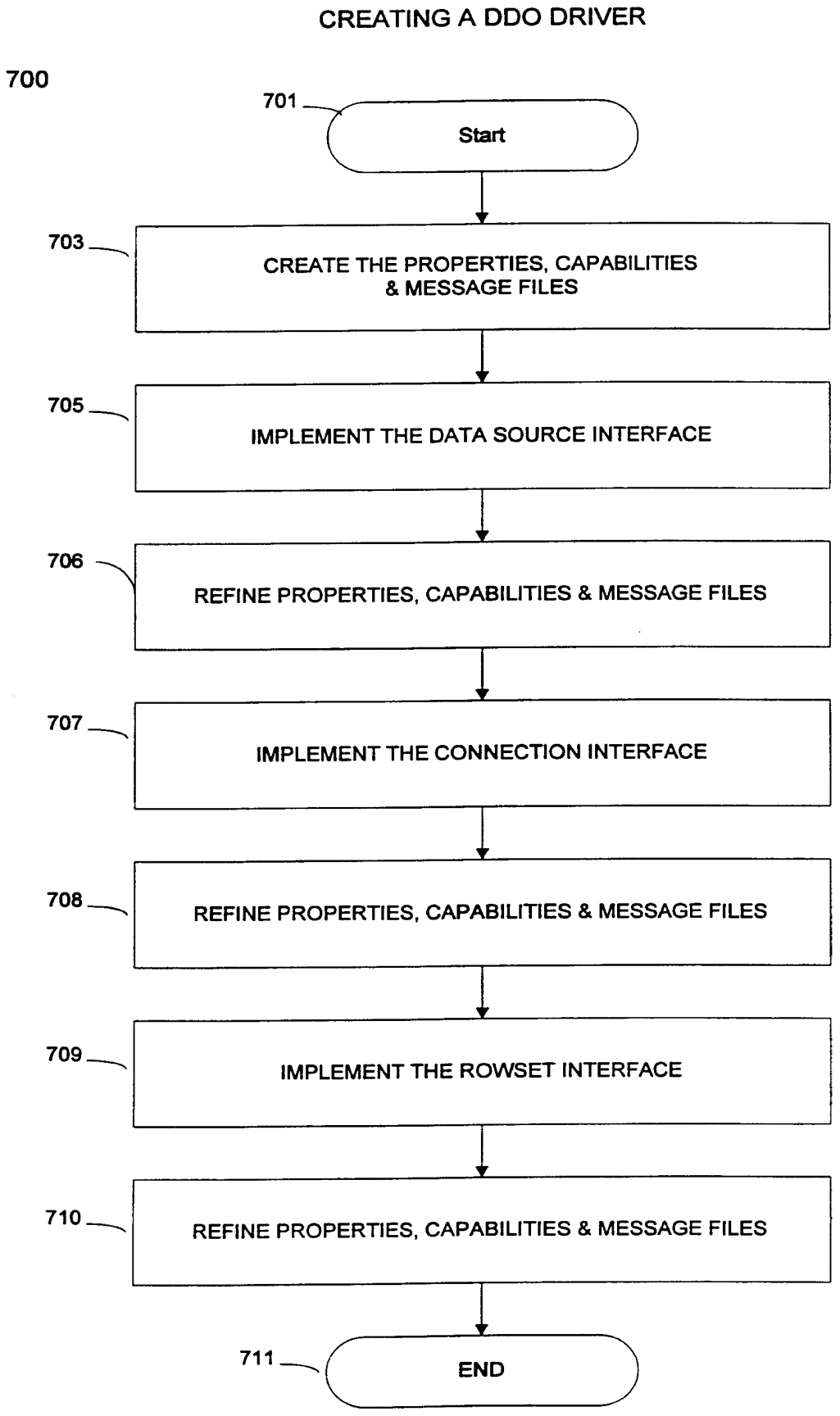

A
625
GET METADATA
627
GET OBJECTS & PROCEDURES
629
EXECUTE COMMANDS
631
RETRIEVE DATA
633
DISPLAY DATA
635
END

FIG. 7


CREATING A DDO DRIVER
700
701
Start
703
CREATE THE PROPERTIES, CAPABILITIES & MESSAGE FILES
705
IMPLEMENT THE DATA SOURCE INTERFACE
706
REFINE PROPERTIES, CAPABILITIES & MESSAGE FILES
707
IMPLEMENT THE CONNECTION INTERFACE
708
REFINE PROPERTIES, CAPABILITIES & MESSAGE FILES
709
IMPLEMENT THE ROWSET INTERFACE
710
REFINE PROPERTIES, CAPABILITIES & MESSAGE FILES
711
END

(12) INTERNATIONAL APPLICATION PUBLISHED UNDER THE PATENT COOPERATION TREATY (PCT)

(19) World Intellectual Property Organization
International Bureau

(43) International Publication Date
14 December 2000 (14.12.2000)

PCT

(10) International Publication Number
WO 00/75849 A3

(51) International Patent Classification[7]: G06F 17/60, 17/30

(21) International Application Number: PCT/US00/04249

(22) International Filing Date: 18 February 2000 (18.02.2000)

(25) Filing Language: English

(26) Publication Language: English

(30) Priority Data:
09/328,049 8 June 1999 (08.06.1999) US

(63) Related by continuation (CON) or continuation-in-part (CIP) to earlier application:
US 09/328,049 (CIP)
Filed on 8 June 1999 (08.06.1999)

(71) Applicant *(for all designated States except US)*: **BRIO TECHNOLOGY, INC.** [US/US]; 3460 West Bayshore Road, Palo Alto, CA 94303 (US).

(72) Inventors; and
(75) Inventors/Applicants *(for US only)*: **LACKEY, Richard, L.** [US/US]; 16901 Spencer Avenue, Los Gatos, CA 95032 (US). **YEDWAB, Gadi** [US/US]; 4256 Fir Avenue, Seal Beach, CA 90740 (US).

(74) Agents: **BASINSKI, Erwin, J.** et al.; Morrison & Foerster LLP, 755 Page Mill Road, Palo Alto, CA 94304-1018 (US).

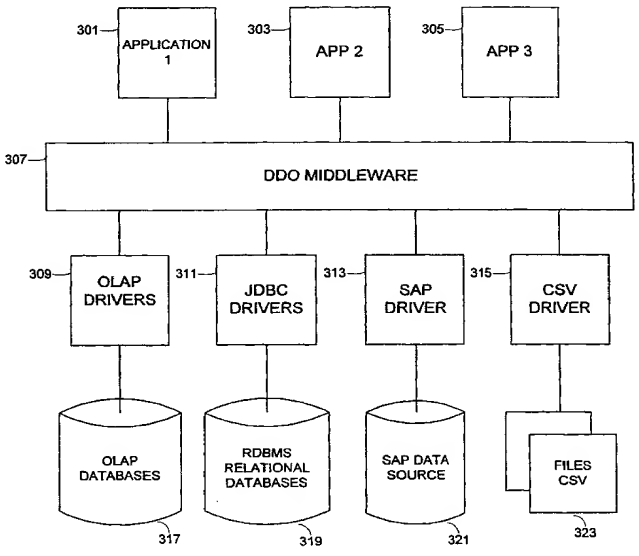
(81) Designated States *(national)*: AL, AM, AT, AU, AZ, BA, BB, BG, BR, BY, CA, CH, CN, CR, CU, CZ, DE, DK, DM, EE, ES, FI, GB, GD, GE, GH, GM, HR, HU, ID, IL, IN, IS, JP, KE, KG, KP, KR, KZ, LC, LK, LR, LS, LT, LU, LV, MA, MD, MG, MK, MN, MW, MX, NO, NZ, PL, PT, RO, RU, SD, SE, SG, SI, SK, SL, TJ, TM, TR, TT, TZ, UA, UG, US, UZ, VN, YU, ZA, ZW.

(84) Designated States *(regional)*: ARIPO patent (GH, GM, KE, LS, MW, SD, SL, SZ, TZ, UG, ZW), Eurasian patent (AM, AZ, BY, KG, KZ, MD, RU, TJ, TM), European patent (AT, BE, CH, CY, DE, DK, ES, FI, FR, GB, GR, IE, IT, LU, MC, NL, PT, SE), OAPI patent (BF, BJ, CF, CG, CI, CM, GA, GN, GW, ML, MR, NE, SN, TD, TG).

Published:
— *with international search report*

[Continued on next page]

(54) Title: METHOD AND APPARATUS FOR DATA ACCESS TO HETEROGENEOUS DATA SOURCES



(57) Abstract: The present invention is a middleware system which can provided efficient access to disparate data sources such as relational databases, non-relational databases, multidimensional databases and the like, in a manner which requires the selection of the data access parameters to be done only once and wherein these selected parameters are thereafter useable for the desired data access regardless of changes to the file structures of the data sources themselves. Additionally, access to newly specified data sources can be easily added to the system. Obtained data from the disparate data sources are displayed in a common format regardless of the source of the data and are displaced as streamed result sets.

WO 00/75849 A3

(88) Date of publication of the international search report:
21 March 2002

For two-letter codes and other abbreviations, refer to the "Guidance Notes on Codes and Abbreviations" appearing at the beginning of each regular issue of the PCT Gazette.

INTERNATIONAL SEARCH REPORT

International Application No
PCT/US 00/04249

A. CLASSIFICATION OF SUBJECT MATTER
IPC 7 G06F17/60 G06F17/30

According to International Patent Classification (IPC) or to both national classification and IPC

B. FIELDS SEARCHED

Minimum documentation searched (classification system followed by classification symbols)
IPC 7 G06F

Documentation searched other than minimum documentation to the extent that such documents are included in the fields searched

Electronic data base consulted during the international search (name of data base and, where practical, search terms used)
EPO-Internal, WPI Data, PAJ, INSPEC, IBM-TDB

C. DOCUMENTS CONSIDERED TO BE RELEVANT

Category °	Citation of document, with indication, where appropriate, of the relevant passages	Relevant to claim No.
X	HAMMER J ET AL: "Template-based wrappers in the TSIMMIS system" SIGMOD RECORD, SIGMOD, NEW YORK, NY, US, vol. 26, no. 2, 13 May 1997 (1997-05-13), pages 532-535, XP002142759 ISSN: 0163-5808 the whole document	1-29
X	US 5 634 053 A (NOBLE WILLIAM B ET AL) 27 May 1997 (1997-05-27)	1-3,6, 9-11,14, 19,21, 25,26
A	column 3, line 8 - line 45; figure 1 column 5, line 10 -column 8, line 40	24,28,29

-/--

[X] Further documents are listed in the continuation of box C.

[X] Patent family members are listed in annex.

° Special categories of cited documents :

"A" document defining the general state of the art which is not considered to be of particular relevance

"E" earlier document but published on or after the international filing date

"L" document which may throw doubts on priority claim(s) or which is cited to establish the publication date of another citation or other special reason (as specified)

"O" document referring to an oral disclosure, use, exhibition or other means

"P" document published prior to the international filing date but later than the priority date claimed

"T" later document published after the international filing date or priority date and not in conflict with the application but cited to understand the principle or theory underlying the invention

"X" document of particular relevance; the claimed invention cannot be considered novel or cannot be considered to involve an inventive step when the document is taken alone

"Y" document of particular relevance; the claimed invention cannot be considered to involve an inventive step when the document is combined with one or more other such documents, such combination being obvious to a person skilled in the art.

"&" document member of the same patent family

Date of the actual completion of the international search	Date of mailing of the international search report
4 December 2001	11/12/2001

Name and mailing address of the ISA	Authorized officer
European Patent Office, P.B. 5818 Patentlaan 2 NL - 2280 HV Rijswijk Tel. (+31-70) 340-2040, Tx. 31 651 epo nl, Fax: (+31-70) 340-3016	Deane, E

1

Form PCT/ISA/210 (second sheet) (July 1992)

INTERNATIONAL SEARCH REPORT

International Application No
PCT/US 00/04249

C.(Continuation) DOCUMENTS CONSIDERED TO BE RELEVANT

Category °	Citation of document, with indication, where appropriate, of the relevant passages	Relevant to claim No.
X	EP 0 625 756 A (HUGHES AIRCRAFT CO) 23 November 1994 (1994-11-23) abstract; figure 1 column 7, line 12 -column 8, line 2	1,14
X	PAPAKONSTANTINOU Y ET AL: "A query translation scheme for rapid implementation of wrappers" DEDUCTIVE AND OBJECT-ORIENTED DATABASES. PROCEEDINGS OF THE INTERNATIONAL CONFERENCE, XX, XX, 4 December 1995 (1995-12-04), pages 1-26-186, XP002115326 abstract page 1, line 1 -page 4, line 28; figure 1	1,14,19, 24,25, 28,29

Form PCT/ISA/210 (continuation of second sheet) (July 1992)

INTERNATIONAL SEARCH REPORT

Information on patent family members

In[illegible]national Application No
PCT/US 00/04249

Patent document cited in search report	Publication date	Patent family member(s)	Publication date
US 5634053 A	27-05-1997	NONE	
EP 0625756 A	23-11-1994	US 5596744 A	21-01-1997
		CA 2123822 A1	21-11-1994
		DE 69422657 D1	24-02-2000
		DE 69422657 T2	31-08-2000
		EP 0625756 A1	23-11-1994
		JP 7141399 A	02-06-1995